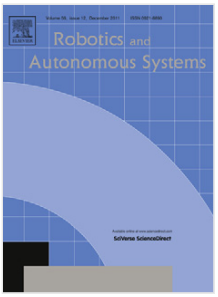
Accepted Manuscript



Direct off-line robot programming via a common CAD package

Pedro Neto, Nuno Mendes

PII: S0921-8890(13)00041-9
DOI: http://dx.doi.org/10.1016/j.robot.2013.02.005
Reference: ROBOT 2106

To appear in: *Robotics and Autonomous Systems*

Received date: 13 July 2012
Revised date: 30 January 2013
Accepted date: 8 February 2013

Please cite this article as: P. Neto, N. Mendes, Direct off-line robot programming via a common CAD package, *Robotics and Autonomous Systems* (2013), http://dx.doi.org/10.1016/j.robot.2013.02.005

Direct off-line robot programming via a common CAD package

Pedro Neto, Nuno Mendes

CEMUC, Department of Mechanical Engineering - POLO II, University of Coimbra, 3030-708 Coimbra, Portugal

Corresponding author: Pedro Neto

Email: pedro.neto@dem.uc.pt

Tel: +351 239 790 700

Abstract:

This paper focuses on intuitive and direct off-line robot programming from a CAD drawing running on a common 3-D CAD package. It explores the most suitable way to represent robot motion in a CAD drawing, how to automatically extract such motion data from the drawing, make the mapping of data from the virtual (CAD model) to the real environment and the process of automatic generation of robot paths/programs. In summary, this study aims to present a novel CAD-based robot programming system accessible to anyone with basic knowledge of CAD and robotics. Experiments on different manipulation tasks show the effectiveness and versatility of the proposed approach.

Keywords: Intuitive Robot Programming, CAD, Off-Line Programming

1. Introduction

Robot programming through the conventional teaching process (using the teach pendant) is often a tedious and time-consuming task that demands significant technical expertise. Many companies, especially small and medium-sized enterprises (SMEs), are not using robots and/or other automatic systems in their facilities because the configuration and programming process of this type of equipments is time-consuming and requires workers with knowledge in the field [1]. Nevertheless, most industrial robots are still programmed using the conventional teaching process. Thus, new and more intuitive approaches to robot programming are required. In fact, teach pendants are not intuitive to use [2-5] and some authors have presented solutions to this problem. This may involve the introduction of mechanisms for collision avoidance and automatic path planning in the robot teaching process [3, 4]. Off-line robot programming (OLP) has increased in popularity over the years, with advantages and disadvantages over lead-through methods (see section 2) [6-8].

Drawing inspiration from the way humans communicate with each other, this paper explores and studies methodologies that can help robot users to interact with a robot in an intuitive way, with a high-level of abstraction from the robot specific language. In fact, a human being can be taught in several different ways, for example, through drawings. As an example, it is very common to see a human being explaining something to another human being with base on a CAD drawing. In practice, CAD data have been used in robot programming with some degree of reliability since the 80's, see section 2.1.

In this paper, we present a novel system for CAD-based OLP, Fig. 1. Robot programs are directly generated from a 3-D CAD drawing running on a commonly available 3-D CAD package and not from commercial OLP or CAM software. The aim is to automatically generate robot motion sequences (programs) from a graphical description of the robot paths over a 3-D CAD model of a given robotic cell. A unified treatment of CAD and robot programming methods may involve very important advances in versatility and autonomy of the platform, in other words, product design and robot programming can be integrated seamlessly. It is explored the most suitable way to represent robot motion in a CAD drawing, how to automatically extract such motion data from the drawing, make the mapping of data from the virtual (CAD model) to the real environment and the process of automatic generation of robot paths/programs. A major goal is to create a CAD-based OLP system accessible to anyone with basic knowledge of CAD and robotics. Since today's CAD packages are rather widespread, are relatively easy to use and have affordable prices, this can open the door to new robot users and thus contribute to increase the number of existing robots in companies. Some algorithms with running code are presented, allowing readers to replicate and improve the work done so far. Experiments on different manipulation tasks show the effectiveness and versatility of the proposed approach.

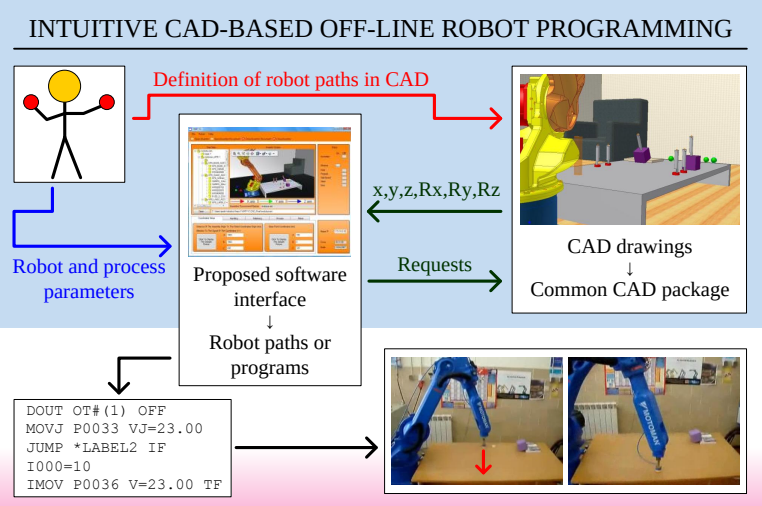


Fig. 1. Overview of the proposed approach.

2. Off-line robot programming

OLP is not a "fully automatic" programming process, it may involve manual editing of robot code and/or the definition of the robot programs by means of computer software that simulates the real robotic scenario. Some major advantages of OLP include:

- Robot programming without stopping/disturbing robot production, Fig. 2. Robots can be programmed before installation and stay in production while being re-programmed for a new task [6]. This means that robot programming can be carried out in parallel with robot production (production breaks are shortened);
- The programming efforts are moved from the robot operator in the workshop (factory floor) to the engineer/programmer in the office;
- Increase of work safety. During the programming process the user is not in the robot working area;
- Robot programs can be tested using simulation tools. This is very important to anticipate the real robots behaviour and in this way to optimize working processes.

On the other hand, some disadvantages can be pointed out:

- Relatively high initial investment in software and workers' training. This investment is difficult to justify for most SMEs;
- Error associated with robot calibration. Robot calibration requires highly expensive measurement hardware, software and technical knowledge;
- The task calibration process requires experienced operators. A rough task calibration can lead to tremendous inaccuracies during robot operation;
- Robot programs created off-line need to be tested in the real robot in order to verify if they run correctly. In this context, calibration errors can lead to robot crashes.
- Process information is required in advance;
- OLP methods rely on accurate modelling of the robotic cell.

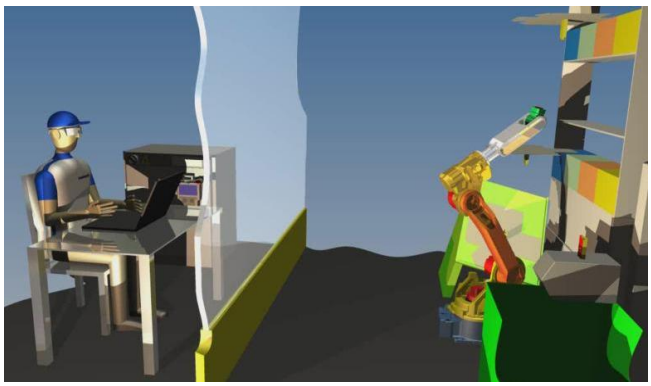


Fig. 2. OLP concept.

Software packages dedicated to OLP are usually called OLP software or computer-aided robotics (CAR) software. Some OLP packages are able to operate with robots from different manufacturers (generic OLP packages). Three of the most common generic OLP packages are *Delmia* from *Dassault Systèmes*, *RobCAD* from *Technomatix Technologies* and *Robotmaster* from *Jabez Technologies*. These software packages provide a set of modelling and simulation tools capable to represent graphically a robot manipulator and its attendant equipment, generate programs, and hence simulate a given robotic task [7, 8]. On the other hand, almost every robot manufacturer has its own OLP software. Examples are *KUKA.Sim* from *KUKA*, *RobotStudio* from *ABB Robotics* and *MotoSim* from *Motoman*. Early versions of OLP software were based on simple wireframe models of the robot's kinematics. However, in recent years, robot simulation techniques have seen a rise in realism and popularity, possibly coinciding with the advancement of computing and graphical animation technologies. OLP packages of today are more graphically powerful, modular (with modules for specific processes such as coating and welding) and standard (with capacity for example to import standard CAD formats).

All of these capabilities come at a cost. A license for OLP software can costs thousands of Euros, an investment difficult to justify for most SMEs. Advantages of OLP software are tempered by some limitations in existing systems. In fact, they are not intuitive to use and can only be applied in situations where the robot surrounding environment is known a priori and well modelled [9]. In addition, high absolute positioning accuracy can only be achieved with correctly calibrated robots [10-12]. If the robot poses (positions and orientations) are manually taught, repeatability is an important factor and positioning accuracy is not [12]. On the contrary, in OLP, positioning accuracy is a factor of crucial importance because robot paths are defined in a virtual space with respect to a given coordinate system. The positioning accuracy of an industrial robot varies with the manufacturer, age and robot type. The error magnitude can be as low as a tenth of a millimetre or as high as several centimetres. An appropriate calibration can reduce it to less than a millimetre. The international standard ISO 9283 recommends procedures for a correct calibration process. Different hardware and techniques have been applied in robot calibration, for example the ROSY system that uses a calibration ball and digital cameras to calculate kinematic errors and the resulting correction values (compensatory parameters) [10]. Another study shows how the accuracy of an ABB IRB 1600 industrial robot is improved using a 29-parameter calibration model [11]. Measures are acquired with a laser tracker. Most robot manufacturers provide robot calibration services.

2.1. CAD-based robot programming

In recent years, CAD technology has become economically attractive and easy to work with. Today, millions of SMEs worldwide are using CAD technology to design and model their products. Nevertheless, the CAD industry has to face significant technical challenges in future [13].

Already in the 80's, CAD was seen as a technology that could help in the development of robotics [14]. Since then, a variety of research has been conducted in the field of CAD-based robot planning and programming. Over the years some researchers have explored CAD technology trying to extend its capabilities to the robotics field. Today, it is possible to extract information from CAD drawings/files to generate robot paths/programs for many different applications [15-18].

A series of studies have been conducted using CAD as an interface between robots and humans. Diverse solutions have been proposed for the processes of spray painting and coating. A review of CAD-based robot path planning for spray painting is presented by Chen *et al*. [19]. A CAD-guided robot path generator is proposed for the process of spray painting of compound surfaces commonly seen in automotive manufacturing [20]. Arikan and Balkan propose a CAD-based robotic system addressing the spray painting process of curved surfaces (OLP and simulation) [21] and Chen *et al.* a CAD-based automated robot trajectory planning system for spray painting of free-form surfaces [22].

An important study in the field of CAD-based robotics presents a method to generate 3-D robot working paths for a robotic adhesive spray system for shoe outsoles and uppers [23]. An example of a novel process that benefits from robots and CAD versatility is the so-called incremental forming process of metal sheets. Without using any costly form, metal sheets are clamped in a rigid frame and the robot produces a given 3-D contour by guiding a tool equipped with a high-frequency oscillating stamp over the metal surface. The robot's trajectories are computed from the CAD model on the basis of specific material models. Prototype panels or customized car panels can be economically produced using this method [24]. Pulkkinen *et al.* present a robot programming concept for applications where metal profiles are processed by robots and only a 2-D geometrical representation of the workpiece is available [25].

Nagata *et al*. propose a robotic sanding platform where robot paths are generated by CAD/CAM software [26]. A robotic CAD/CAM system that allows industrial robots to move along CL data without using any robot language is presented by Nagata *et al*. [27, 28]. A recent study discusses robot path generation from CAM software for rapid prototyping applications [29]. Feng-yun and Tian-sheng present a robot path generator for a polishing process where CL data are generated from the postprocessor of a CAD system [30]. Other previous studies report the development of robotic systems for rapid prototyping in which cutting data are extracted from CAD drawings [31-32]. A CAD-based system to generate deburring paths from CAD data is proposed by Murphy *et al.* [33]. A method for manufacturing prototype castings using a robot-based system in which manufacturing paths are generated from CAM software is proposed by Sallinen and Servio [34]. In a different kind of application, CAD drawings are used for robot navigation purposes, namely for large scale path planning [35].

As we have seen above, a variety of research has been conducted in the fields of CAD-, CAM- and VRML-based OLP. However, none of the studies so far has an effective solution for an intuitive and low-cost OLP solution using raw CAD data and directly interfacing with a commercial CAD package.

Research studies in this area have produced great results, some of them already implemented in industry, but limited to a specific industrial process (welding, painting, etc.). Even though a variety of approaches has been presented, a cost-effective and standard solution has not been established yet.

3. CAD-based approach

3.1. CAD packages

CAD technology has become economically attractive and easy to work with so that today there are millions of companies worldwide using it to design and model their products. While the prices of CAD packages have decreased, their features and functionalities have been upgraded, with improved and simplified user interfaces, user-oriented functionalities, automatic design of standard products, etc. Nowadays, most CAD packages provide a wide range of associated features (integrated modules or standalone solutions) that not only help in the effective design process, but also help in other tasks such as mechanical simulation and the physical simulation of dynamic processes. Robot programming and simulation has been seen as another feature that CAD packages can integrate.

Autodesk Inventor, which is one of the most common 3-D CAD packages of today, was chosen to serve as interface with the proposed solution. It incorporates all the functionalities of modern CAD packages: design, visualization, simulation, user-friendly interface and provides a complete application programming interface (API) for customization purposes, allowing developers to customize their CAD-based applications [36]. In terms of file formats, besides all the standard formats, *Autodesk Inventor* has proprietary file formats to define single *part model* files (ipt file) and *assembly model* files (iam file).

3.2. Extracting data from CAD drawings

The base of the proposed CAD-based OLP platform is the ability to automatically extract robot motion data from CAD drawings running on *Autodesk Inventor*. The *Autodesk Inventor API* is used for that purpose. It exposes the *Inventor's* functionalities in an object-oriented manner using a technology from *Microsoft* called *Automation*. In this way, developers can interact with *Autodesk Inventor* using current programming languages such as Visual Basic (VB), Visual C# and Visual C++. The *API* allows developers to create a software interface that performs the same type (kind) of operations that a user can perform when using *Autodesk Inventor* interactively. Summarizing, the *API* provides a set of routines that may be used to build a software interface based on resources from *Autodesk Inventor*.

There are different ways to access the *Autodesk Inventor API*, Fig. 3. The white boxes represent components provided by the *API* (*Autodesk Inventor* and *Apprentice Server*) and the gray boxes represent programs written by developers. When one box encloses another box, this is an indication that the enclosed box is running in the same process as the box which is enclosing it. Thus, an "in-process" program will run significantly faster than a program running out of the process.

In the context of this paper, a standalone application is proposed to access the *API* and subsequently *Inventor* data. This choice was due to the necessity to integrate in the main application not only the process of interaction with CAD but also other software components for other tasks, for example, robot communications.

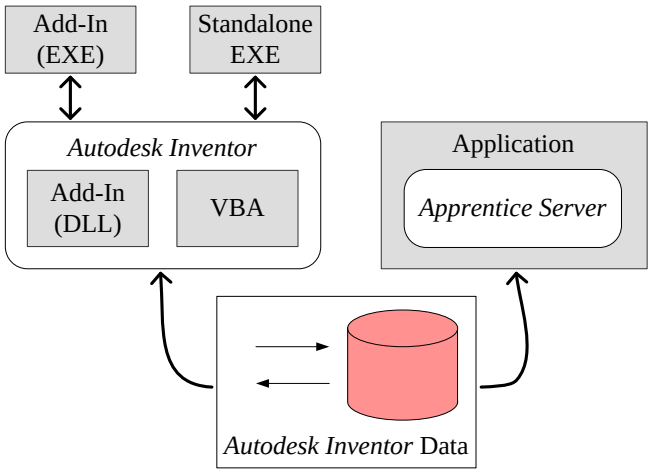


Fig. 3. Accessing the Autodesk Inventor's API.

The *API* provided by *Autodesk* has a number of functionalities that were explored to be used in robotics. As an example, it is possible through a standalone application to open *Autodesk Inventor* in visible mode, Fig. 4, and open an *Inventor* document, Fig. 5. The properties of the document can then be easily accessed, Fig. 5.

Algorithm 1. Opening *Autodesk Inventor* (coded in VB).

```
1  ' Get object.
2  Private oApp As Inventor.Application
3  Try
4      oApp =
       System.Runtime.InteropServices.Marshal.GetActiveObject("Inventor.Application")
5  Catch ex As Exception
6      MessageBox.Show("A problem occurs")
7  End Try
8  ' Open Inventor (oApp).
9  Dim InventorAppType As Type = System.Type.GetTypeFromProgID("Inventor.Application")
10 oApp = System.Activator.CreateInstance(InventorAppType)
11 ' Make Inventor visible.
12 oApp.Visible = True
```

Fig. 4. Opening *Autodesk Inventor* (coded in VB).

Algorithm 2. Opening an *Autodesk Inventor* document (coded in VB).

```
1  ' Open an Inventor document (InventorDoc).
2  Private oApp As Inventor.Application
3  Private InventorDoc As Inventor.Document
4  InventorDoc = oApp.Documents.Open("document name")
5  ' Properties of the document.
6  Dim oPropsets As PropertySets
7  oPropsets = InventorDoc.PropertySets
```

Fig. 5. Opening an *Inventor* document and extracting their properties (coded in VB).

There are a great number of data that can be extracted from a CAD drawing. The question is: what data are required to achieve our goal (OLP)? In practice, we need to have robot motion from CAD

drawings, i.e., a sequence of target points that representing the robot end-effector poses with respect to a known coordinate system (in Cartesian space). Thus, given the capacity of the *Autodesk Inventor API*, it was established that we need to extract positions and orientations of objects in 3-D space from proper CAD drawings representing a given robotic cell:

1. Positions - positional data can be acquired from a CAD drawing in different ways, for example, acquiring *WorkPoints* positional data (points that can be placed in the CAD drawing – see section 3.3), Fig. 6. In other situations, positional data come from the points that characterize each one of the different lines representing virtual robot paths in a CAD drawing, Fig. 7. For example, in a specific situation, if the robot paths assume the geometry of a spline in the CAD drawing, the *API* provides all the points necessary to define such geometry. All these data are defined in relation to the origin of the CAD *assembly model* of the robotic cell;

2. Orientations - the *API* provides information about the transformation matrix (or homogeneous transform) of each *part model* represented in a CAD *assembly model*, Fig. 8. The transformation matrix contains the rotation matrix and the position of the origin of the *part model* to which it refers, both in relation to the origin of the CAD *assembly model* of the robotic cell.

Algorithm 3. Extracting data from a selected *WorkPoint* (coded in VB).

```
1  ' Select an item.
2  Dim oSelectSet As SelectSet
3  oSelectSet = ThisApplication.ActiveDocument.SelectSet
4  ' Check if the selected item is a WorkPoint.
5  If TypeOf oSelectSet.Item(i) Is WorkPoint Then
6  |   Dim wp1(i) As WorkPoint
7  |   wp1(i) = oSelectSet.Item(i)
8  |   ' WorkPoint data (name, X, Y, Z).
9  |   WorkPointPos(i).oName = wp1(i).Name
10 |   WorkPointPos(i).x = wp1(i).Point.X
11 |   WorkPointPos(i).y = wp1(i).Point.Y
12 |   WorkPointPos(i).z = wp1(i).Point.Z
13 Else
14 |   MsgBox("You must select a WorkPoint.")
15 |   Exit Sub
16 End If
```

Fig. 6. Extracting data from a selected *WorkPoint* (coded in VB).

Algorithm 4. Extracting data from a selected virtual line (coded in VB).

```
'Extracting straight line data.
If TypeOf oSelectSet.Item(i) Is SketchLine3D Then
    'Defining an object type SketchLine3D.
    Dim Line_ As SketchLine3D
    Line_ = oSelectSet.Item(i)
    'Start and end points of the SketchLine3D.
    Dim start_point_x, start_point_y, start_point_z As Double
    Dim end_point_x, end_point_y, end_point_z As Double
    start_point_x = Line_.StartSketchPoint.Geometry.X 'The same for y and z.
    end_point_x = Line_.EndSketchPoint.Geometry.X 'The same for y and z.
    'Extracting spline data.
ElseIf TypeOf oSelectSet.Item(i) Is SketchSpline3D Then
    'Defining an object type SketchSpline3D.
    Dim Spline_ As SketchSpline3D
    Spline_ = oSelectSet.Item(i)
    'Start, medium and end points of the SketchSpline3D.
    Dim s_start_point_x, s_start_point_y, s_start_point_z As Double
    Dim s_mid_point_x, s_mid_point_y, s_mid_point_z As Double
    Dim s_end_point_x, s_end_point_y, s_end_point_z As Double
    s_start_point_x = Spline_.StartSketchPoint.Geometry.X 'The same for y and z.
    s_mid_point_x = Spline_.FitPoint(2).Geometry.X 'The same for y and z.
    s_end_point_x = Spline_.EndSketchPoint.Geometry.X 'The same for y and z.
End If
```

Fig. 7. Extracting data from a selected virtual line (coded in VB).

Algorithm 5. Transformation matrix (coded in VB).

```
' Get an occurence from the selected item.
Dim oOccurrence As ComponentOccurrence
oOccurrence = ThisApplication.ActiveDocument.SelectSet.Item(i)
' Get the transformation matrix.
Dim oTransform As Inventor.Matrix
oTransform = oOccurrence.Transformation
' Get matrix data, for example cell (1, 3).
Dim mt13 As Double
mt13 = oTransform.Cell(1, 3)
```

Fig. 8. Extracting the transformation matrix of a selected item (coded in VB).

3.3. CAD models

The process of creating the CAD *part models* that compose the CAD *assembly model* of the robotic cell should respect some rules. Since it was previously established that we need to have represented in the CAD *assembly model* of the robotic cell all the required robot paths (end-effector poses), it becomes necessary to study the most suitable way to have that information represented in CAD drawings. This can be achieved in two different ways:

1. Introducing extra robot tool (end-effector) models within the *assembly model*. These models represent the desired robot end-effector pose in each segment of the path, Fig. 9. Positional data are achieved by placing a *WorkPoint* attached to any part of the tool model, Fig. 10. The *WorkPoint* data (x,y,z) are provided by the *API* in relation to the origin of the CAD *assembly model*. Orientation data are achieved from the tool models orientation in the drawing (transformation matrix);

2. Drawing lines (in the *assembly model*) representing the desired robot path (positional data) and defining the robot end-effector orientation by placing simplified tool models along the path lines (in each segment of the path), as in the above topic, Fig. 9.

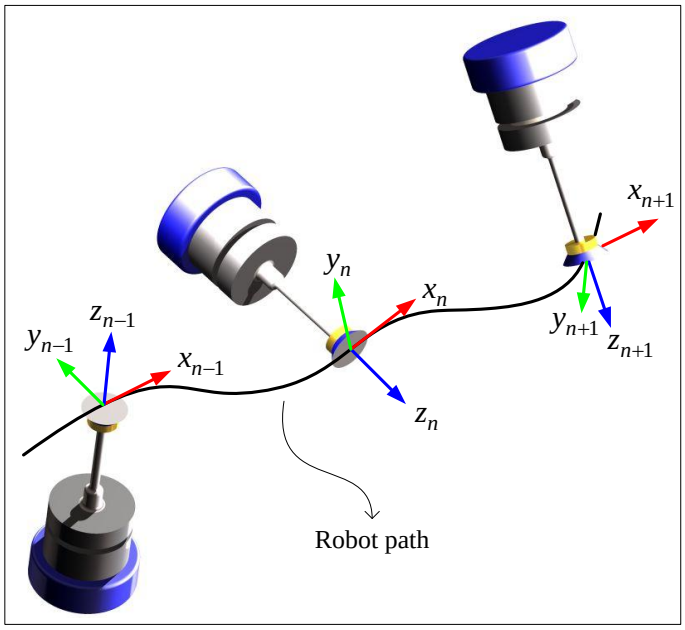


Fig. 9. Simplified tool models defining the robot end-effector pose in each path segment.

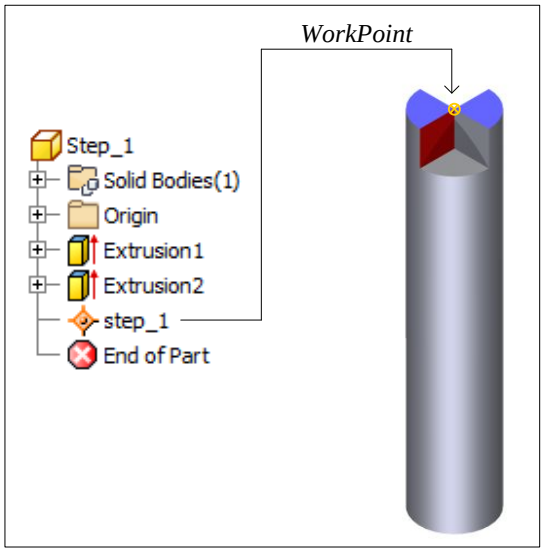


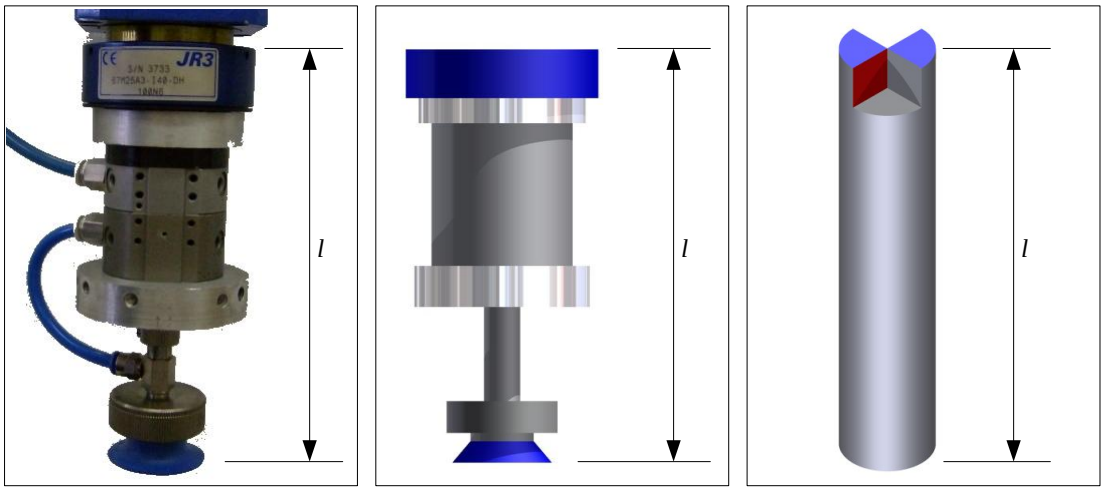
Fig. 10. A *WorkPoint* attached to a tool model in a location where the tool is connected to the robot wrist.

The CAD *assembly model* does not need to accurately represent the real cell in all its aspects. On the contrary, it can be a simplified model containing all the necessary/important information for the programming process (target points and relations between them). As an example, the robot tool length, robot path positions and relative positioning of CAD models have to accurately represent the real environment. However, the models appearance does not need to be exactly the same as the real objects. It means that, for example, chamfers or rounded edges are expendable. These simplifications allow to speed

up the modelling process. Fig. 11 shows a real robot tool (a) and two CAD models of that tool, (b) and (c), with the same length *l*. These two models were created with different levels of detail:

1. Model (b) was created with more detail than model (c). It represents more accurately the real tool, with advantages in terms of visualization. Nevertheless, the process of drawing this model is more time consuming than drawing model (c);
2. Model (c) is a simpler version but accurate at the same time in terms of total length of the tool. It can be drawn in seconds and used where only the length of the tool is a factor of importance.

It is important to note that the best model is the simplest model that still serves its purpose.



(a) (b) (c)

Fig. 11. Real robot tool (a), and simplified models (b) and (c).

3.3.1. Process/path planning

The process/path planning task occurs during the construction of the CAD *assembly model*, in which the user is planning in advance the "best" process parameters and paths. Depending on the type and complexity of the process in study, the planning task can include several factors:

1. Models selection/construction and definition of the layout of the cell. Some CAD models can be accessed from libraries provided by manufacturers (robots and other peripheral equipment);
2. Robot motion. Robot motion can be indirectly defined by placing simplified tool models and/or virtual paths within the CAD *assembly model*. The desired movement type (linear, circular or spline) is defined by the geometry of the virtual paths or in the software interface;
3. Operation sequences. Robot operation sequences are defined by the name of the tool models. The first five characters of the name of a tool model should be "step_" (simple robot motion). The sixth character and following should be a number defining the ordering sequence for robot motion. Following the sequence number the tool name can have or not a character type letter indicating a specific robot operation, for example, "step_1A" can indicate robot motion plus the activation of a digital output of the robot;

4. Collisions. Collisions should be predicted by the designer during the creation of the CAD model of the cell. The designer should ensure that there are no collisions between the robot and other objects within the workspace. Fig. 12 shows a CAD drawing with two tool models (initial and target pose) and an obstacle. If the robot end-effector is linearly moved from the initial to the target pose a collision occurs. The designer should anticipate this situation and introduce into the drawing "intermediate" tool models to allow the robot to avoid the obstacle, Fig. 13;

5. Grasping and re-grasping/repositioning. These are common situations in industrial robotics, especially in pure manipulation tasks. Many times, in order to properly perform a task, there is a need for re-grasping or repositioning a given workpiece. Fig. 14 shows a re-grasping process in which a grasping location is changed from an initial pose defined by the tool model step_1 to a target pose defined by step_5. Moreover, during the planning phase, it has to be ensured that the robot is operating with valid tool locations, including valid contact conditions between the gripper and the workpiece. Note that as in Fig. 13 and Fig. 14 some "intermediate" tool models are used to avoid collisions during the re-grasping process.

A robot simulation system can be a valuable help in this planning phase, helping to visualize the robotic process (robot motion, possible collisions and the re-grasping operations) and detect existing robot kinematic singularities or robot joint limits [37].

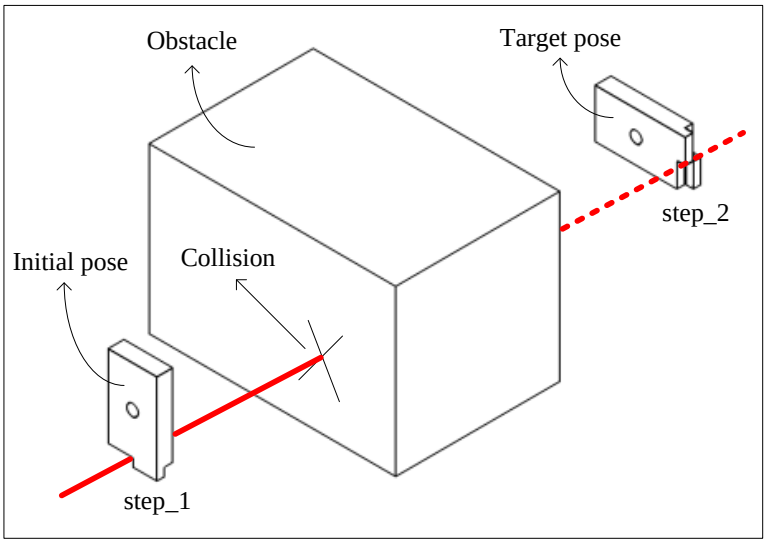


Fig. 12. Collision.

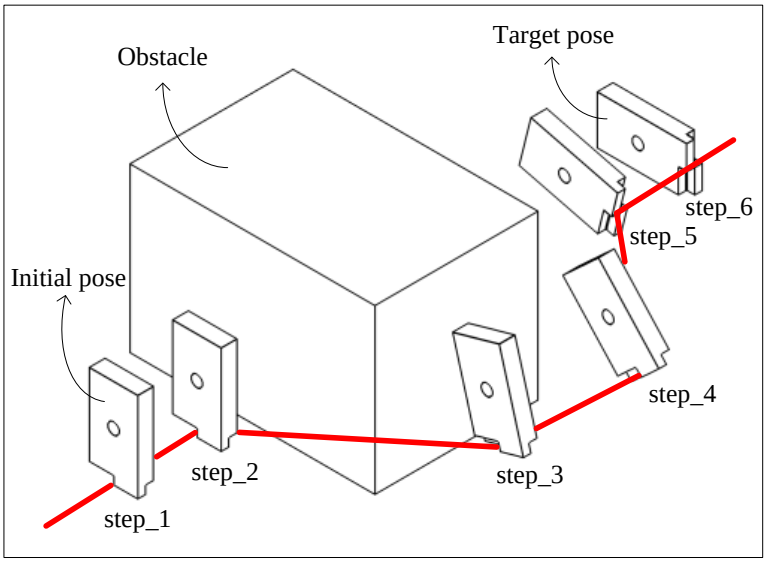


Fig. 13. Avoiding an obstacle.

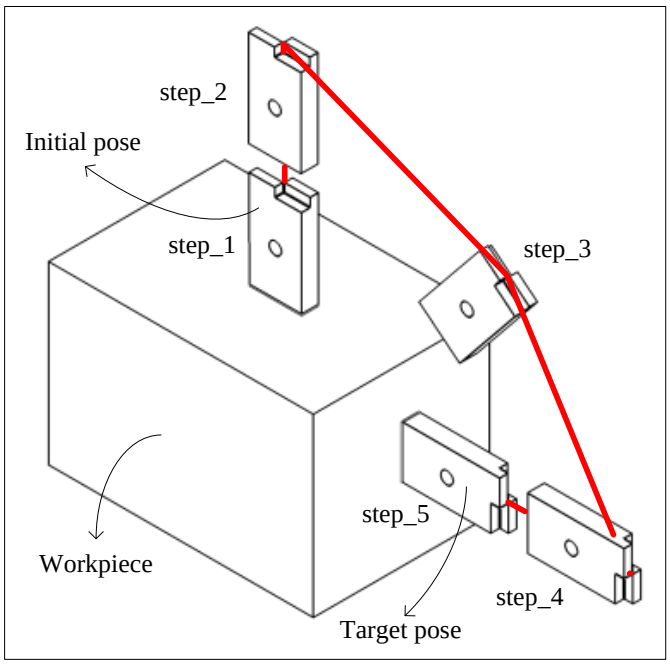


Fig. 14. Re-grasping.

3.4. Mapping and calibration

Many times it is necessary to express the same quantity in relation to different coordinate systems, i.e., change descriptions from frame to frame, mapping. These capabilities can be used for the task calibration process, making the CAD model of the cell in study to match with the real robotic cell. All robot end-effector positions and orientations extracted from CAD have to be known with respect to one or more reference frames known a priori in the space of the real robot. These frames have to be defined within the CAD drawing of the cell by placing invisible *part models* with the desired poses into the CAD *assembly model* (note that each *part model* has a frame associated). Then, the real robot is taught about

that frame(s)' pose in the real environment through the conventional way, using the teach pendant. Essentially, the process consists in the definition of one or more frames within the CAD drawing of the cell and the corresponding frame(s) in the robot controller. This makes the task calibration process a relatively simple and non-time consuming process. Nevertheless, complex robotic scenarios can require the definition of a significant number of different frames. In this case, the task calibration process can be lengthy and prone to error. This is because the user has to remember the pose of each frame previously defined within the CAD drawing and at the same time to define such frames in the real scenario.

As mentioned before, the *Autodesk Inventor API* provides all the information (transformation matrices, *WorkPoints* and path lines data) with respect to the origin of the CAD *assembly model*, here defined by frame {U}, Fig. 15. Frame {B} is defined in the robot controller during the calibration process (in the real robot), and at the same time the *API* provides the transformation matrix of {B} relative to {U}, ${}^{U}_{B}\mathbf{T}$. This means that frame {B} "makes the link" between the virtual and real world. Note that, as mentioned above, it is possible to define more than one frame if necessary, as the process is similar.

Since *Autodesk Inventor* considers the tool models (with a *WorkPoint* attached) and the path lines as a constituent of a single *part model* within in the CAD *assembly model*, the transformation matrix (relative to {U}) of that single *part model* defines the pose of tool models and path lines. For the general case presented in Fig. 15 the path line is part of the table top model. The table top model has the origin and orientation defined by {E}. However, it is not necessary to know the orientation of the path lines because the *API* gives all the necessary points to define the path lines relative to {U}, for example the initial path point ${}^{U}\mathbf{P}_{ini}$, Fig. 15. Thus, it is necessary to achieve the path line points relative to frame {B}. The same for the tool models in which we need to have orientations and *WorkPoint* positional data relative to {B}.

The generic tool models that incorporate {C} and {D}, Fig. 15, help to define the end-effector pose in each path segment, as well as the *WorkPoint* positions (if they have a *WorkPoint* attached). The *API* provides the transformation matrix of these models relative to {U}, ${}^{U}_{C}\mathbf{T}$ and ${}^{U}_{D}\mathbf{T}$. Given our purpose (robot programming), we wish to express {C} and {D} in terms of {B}, ${}^{B}_{C}\mathbf{T}$ and ${}^{B}_{D}\mathbf{T}$. For ${}^{B}_{C}\mathbf{T}$ we have that:

$$ {}^{B}_{C}\mathbf{T} = {}^{B}_{U}\mathbf{T} + {}^{U}_{C}\mathbf{T} \tag{1} $$

To find ${}^{B}_{U}\mathbf{T}$, we must compute the rotation matrix that defines frame {U} relative to {B}, ${}^{B}_{U}\mathbf{R}$, and the vector that locates the origin of frame {U} relative to {B}, ${}^{B}\mathbf{P}_{Uorg}$:

$$ {}^{B}_{U}\mathbf{T} = \left[\begin{array}{ccc|c} & {}^{B}_{U}\mathbf{R} & & {}^{B}\mathbf{P}_{Uorg} \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{2} $$

Let's consider a generic vector/point defined in {U}, ${}^{U}\mathbf{P}$. If we wish to express this point in space in terms of frame {B} we must compute:

$$ {}^{B}\mathbf{P} = {}^{B}_{U}\mathbf{R}\,{}^{U}\mathbf{P} + {}^{B}\mathbf{P}_{Uorg} \tag{3} $$

Given the characteristics of a rotation matrix, ${}^{B}_{U}\mathbf{R} = {}^{U}_{B}\mathbf{R}^{T}$, and as we know ${}^{U}_{B}\mathbf{T}$, there follows the computation of ${}^{B}\mathbf{P}_{Uorg}$. From the process of inverting a transform we have that:

$$ {}^{B}_{U}\mathbf{T} = \left[\begin{array}{ccc|c} & {}^{U}_{B}\mathbf{R}^{T} & & -{}^{U}_{B}\mathbf{R}^{T}\,{}^{U}\mathbf{P}_{Borg} \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{4} $$

Thus, from (2) and (4):

$$ {}^{B}\mathbf{P}_{Uorg} = -{}^{U}_{B}\mathbf{R}^{T}\,{}^{U}\mathbf{P}_{Borg} \tag{5} $$

At this stage, from (1) and (4) we can compute ${}^{B}_{C}\mathbf{T}$. The same methodology can be applied to achieve ${}^{B}_{D}\mathbf{T}$ and any other transformation. This means that all positions and orientations extracted from CAD can be referred with respect to the reference frame(s) defined in the real environment at the moment of the calibration process.

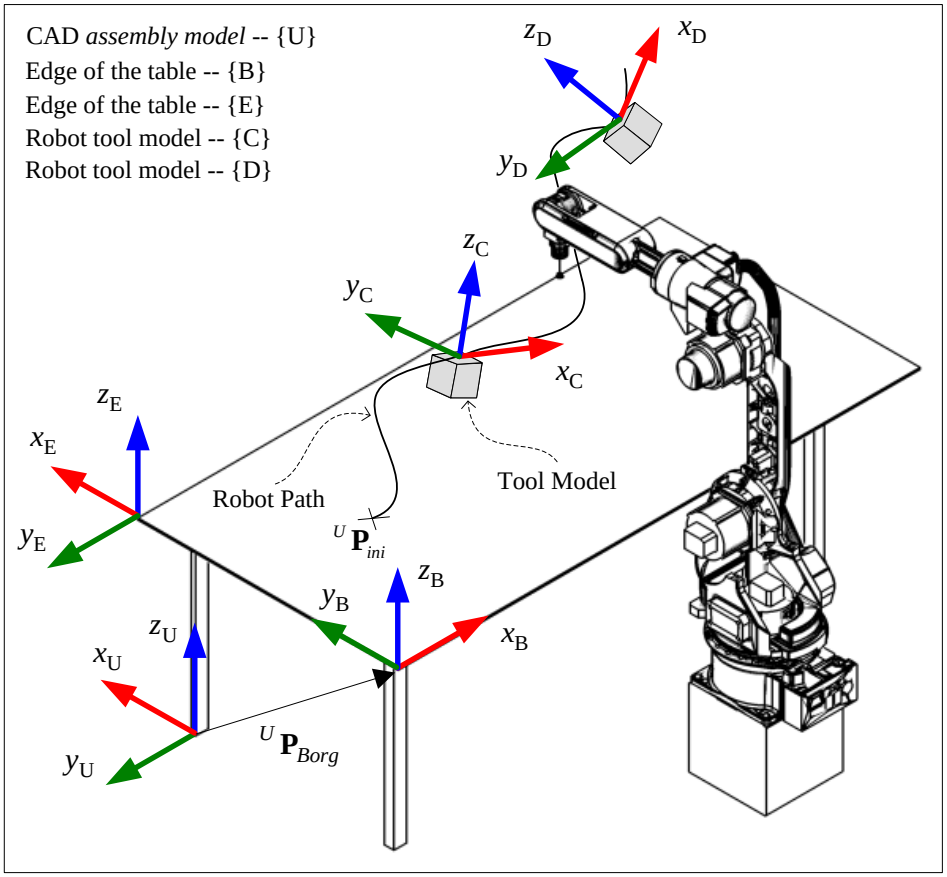


Fig. 15. Coordinate frames.

3.5. X-Y-Z Euler angles

After having obtained from CAD drawings the rotation matrices defining robot end-effector orientations in relation to a given frame, such matrices are transformed into effective end-effector rotations, usually Euler angles or quaternions.

The description of the orientation of a generic frame {B} with respect to a generic frame {A} in the form of X-Y-Z Euler angles (α, β, γ) can be represented by a rotation matrix composed by the multiplication of the rotation matrices around each angle: ${}^{A}_{B}\mathbf{Rot}_{xyz} = \mathbf{Rot}_{x}(\alpha)\,\mathbf{Rot}_{y}(\beta)\,\mathbf{Rot}_{z}(\gamma)$. It is now possible to compute the X-Y-Z Euler angles. If $\beta \neq \pm(\pi/2)$:

$$\beta = \text{Atan2}\left(r_{1,3}, \sqrt{r_{1,1}^2 + r_{1,2}^2}\right) \tag{6}$$

$$\alpha = \text{Atan2}\left(\frac{r_{2,3}}{-\sqrt{r_{1,1}^2 + r_{1,2}^2}}, \frac{r_{3,3}}{\sqrt{r_{1,1}^2 + r_{1,2}^2}}\right) \tag{7}$$

$$\gamma = \text{Atan2}\left(\frac{r_{1,2}}{-\sqrt{r_{1,1}^2 + r_{1,2}^2}}, \frac{r_{1,1}}{\sqrt{r_{1,1}^2 + r_{1,2}^2}}\right) \tag{8}$$

Where $r_{a,b}$ are the elements of ${}^{A}_{B}\mathbf{Rot}_{xyz}$ and Atan2(y, x) is a two argument arc tangent function. When $\beta = \pm(\pi/2)$, the process to compute Euler angles is more complex. In this situation both the x and z axes are aligned with each other and one degree of freedom is lost. This phenomenon is mathematically unsolvable and is known as gimbal lock. In this scenario, α and γ cannot be calculated separately but together:

$$\alpha \pm \gamma = \text{Atan2}\left(r_{3,2}, r_{2,2}\right) \tag{9}$$

The gimbal lock phenomenon does not make Euler angles "wrong" but makes them unsuited for some practical applications. Some methods have been proposed to deal with the gimbal lock phenomenon, for example, solutions based on the representation of rigid body orientation through quaternions [38]. However, some robot manufacturers force the use of Euler angles so that in these cases the option for quaternions is ruled out. Pollard *et al.* propose to locate regions near gimbal lock and compute a restricted degree of freedom solution within those regions [39]. In practice, a typical approach is to set an angle equal to zero and compute the remaining angle. In this case, if $\beta = (\pi/2)$, and assuming that $\alpha = 0$, we have:

$$\gamma = \text{Atan2}\left(r_{2,1}, -r_{3,1}\right) \tag{10}$$

On contrary, if $\beta = -(\pi/2)$ and assuming that $\alpha = 0$, we have:

$$\gamma = \text{Atan2}\left(r_{2,1}, r_{3,1}\right) \tag{11}$$

3.6. Interpolation for end-effector orientations

When an industrial robot is performing a pre-programmed movement and this one requires abrupt end-effector orientation changes, we must take special care because it can come into a situation where no one has total control over the end-effector orientation. In other words, we have no control over the interpolation made by the robot controller between two given poses. This is particularly true when robot programs are generated off-line. In some situations this could be a major problem, leading to the appearance of defects in the work produced by the robot [40]. The proposed solution to circumvent this problem is based on the implementation of linear smooth interpolation of end-effector positions and orientations [30]. The process involves the following steps:

1. Identification of risk areas. This is achieved by analyzing the CAD drawing of the cell and manually defining those areas in the drawing (abrupt end-effector orientation changes);
2. Discretization of the risk robot path in equally spaced intervals;
3. Computation of end-effector orientations for each interpolated path point. The new path is smoother than the initial, Fig. 16.

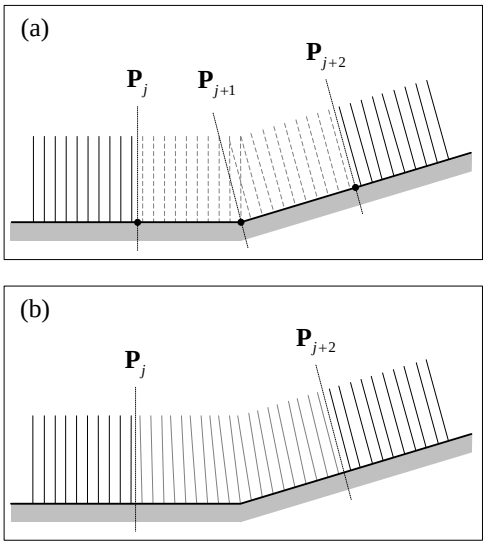


Fig. 16. End-effector poses: (a) before interpolation and (b) after interpolation.

For the profile shown in Fig. 16, interpolation was divided in two sections $S_1 \in \left[\mathbf{P}_j, \mathbf{P}_{j+1}\right]$ and $S_2 \in \left[\mathbf{P}_{j+1}, \mathbf{P}_{j+2}\right]$. The calculations are presented for section S_1 but for other sections the procedure is similar. For a sampling width Δt the interpolated position $\mathbf{r}(k) = (r_x, r_y, r_z)^T$ is:

$$r_i(k) = r_i(0) + v_i(k)\, k\, \Delta t, \quad \begin{cases} (i = x, y, z) \\ k = 1, \ldots, n-1 \end{cases} \tag{12}$$

Where $v_i(k)$ is a directional velocity profile and *n* represents the number of interpolated points.

A spherical linear interpolation (SLERP) algorithm was implemented for the purpose of quaternion interpolation. Given two known unit quaternions, $\mathbf{Q}_0$ (from $\mathbf{P}_j$) and $\mathbf{Q}_n$ (from $\mathbf{P}_{j+1}$) with parameter k moving from 1 to n-1, the interpolated end-effector orientation $\mathbf{Q}_k$ can be obtained as follows:

$$\mathbf{Q}_k = \frac{\sin\left(\left(1-\frac{k-1}{n-1}\right)\theta\right)}{\sin\theta}\mathbf{Q}_0 + \frac{\sin\left(\frac{k-1}{n-1}\theta\right)}{\sin\theta}\mathbf{Q}_n, \quad k = 1,\ldots,n-1 \tag{13}$$

Where:

$$\theta = \cos^{-1}\left(\mathbf{Q}_0 \cdot \mathbf{Q}_n\right) \tag{14}$$

This method for quaternion interpolation is also used when we want to interpolate Euler angles, simply by transforming Euler angles into quaternions and vice versa.

3.7. Generation of robot programs

The search for new and more intuitive methods to programme machines has led to the emergence of techniques to generate machine code. In the last few decades, several code generation techniques have been developed. The most prominent example is the use of commercial CAD/CAM systems to generate reliable CL data for CNC machining [41]. CNC tool paths can also be generated from standard CAD formants [42-44]. Nevertheless, these systems to generate code tend to have some drawbacks such as their ability to generalize from different situations and respond to unseen problems. During the elaboration of an algorithm to generate code, the keyword is “generalize” and never “particularize”, the algorithm must be prepared to cover a wide range of variations in the process. For particular applications with a limited and well known number of process variations this kind of algorithms presents acceptable performance [45].

Robot controller specific languages have seen only minor advances in the last few years. Some authors have devoted attention to create methodologies capable to generalize robot programs around a task but which at the same time can be customized as necessary [46]. An operation can be customized in terms of type of robot operation or shape of the workpiece. Intrinsically, this allows to profit from previously similar work, incorporating the programmers’ experience and process knowledge [47]. Thus, the time to create robot programs for related products/tasks can be reduced and non-specialists can create robot programs by themselves. These systems follow the same logic as the well known macros or scripts in the world of computer science. Translators for robot programming languages have also been matter of concern [48], as well as the development of robotic system that operate without using a specific robot language [27-28].

In this paper, we propose an algorithm to automatically generate robot programs with information extracted from CAD drawings. The way the process to generate robot code is applied differs with the robotic task under study. Nevertheless, there is a common point in all robot programs. It means that since robots usually perform manipulation tasks, the process to generate a robot program does not differ greatly from application to application, containing common tasks like gripping, moving and placing, Fig. 17.

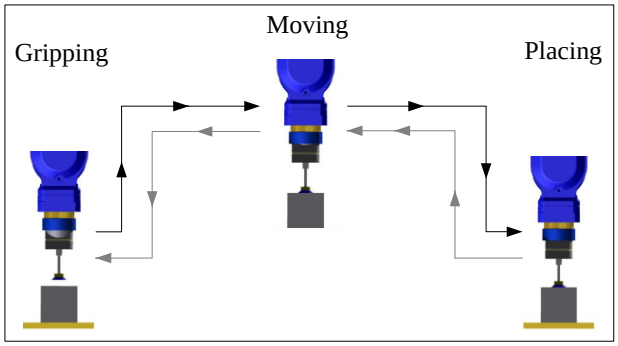


Fig. 17. The different phases of a manipulation task.

The automatic generation of a robot program is no more than writing robot commands in a text file, line by line. In this paper, this process is managed by the software interface (section 3.8) that extracts data from CAD drawings, interprets that data and finally generates robot programs. The process to generate a robot program is divided into two distinct phases:

1. Definition and parameterization of robot end-effector poses, frames, tools and constants. The algorithm in Fig. 18 summarizes the process of data acquisition from tool models and/or path lines and the generation of robot code. The following equation represents a common definition of a robot pose in a robot program.

$$P = x, y, z, \begin{cases} \alpha, \beta, \gamma & \longrightarrow \textit{Euler angles} \\ q_w, q_x, q_y, q_z & \longrightarrow \textit{Quaternions} \end{cases} \tag{15}$$

   In addition to robot poses, specific process and robot parameters (coordinate systems, tools, etc.) are specified in this phase. This information comes from the parameters introduced in the software interface, for example, robot home position, number of working cycles, approaching distances, etc.

2. Body of the program. A robot program contains predominantly robot motion instructions: linear, joint, circular or spline robot movement. These movement instructions respect the type of motion established in the CAD drawing and/or the software interface. For example, if a segment of a path is drawn as a straight line, the generated code will contain a robot instruction that makes the robot end-effector move linearly in that path segment. In this phase the algorithm also has to deal with particular situations associated with each robotic task such as the generation of IO commands to communicate with other machines and the definition of approaching distances.

The proposed algorithm is able to generate robot programs for *Motoman* robot controllers (INFORM language), Fig. 19. However, as all robot programs are based on the same principle, the proposed algorithm can be adapted to generate code in other programming languages.

Algorithm 6 Data extraction from CAD drawings and generation of robot paths/programs

Input: CAD drawing

Output: robot paths/programs

```
Begin
    For Each tool model Do
        Get WorkPoint position, Algorithm 3
        Get transformation matrix, Algorithm 5
        Compute frame correlations, eq (1) to (5)
        Compute Euler angles, eq (6), (7), (8), (10), (11)
        If (tool name = step_n) Then
            Generate code for simple robot motion
        Else If (tool name = step_n...) Then
            Generate code for a given robot operation/task
        End If
    End For
    For Each selected path line in CAD Do
        Get path position, Algorithm 4
        Get end-effector orientation (from tool models)
        Generate robot code
    End For
End
```

Fig. 18. Extracting data from CAD and generation of robot code.

```
...0064=489.3844...
75.050253609301,0,0,0
P00070=489.38445479391,166.36209915648,
775.050253609301,0,0,0
//INST
///DATE 2010/06/29 16:28
///ATTR SC,RW
///GROUP1 RB1
NOP
SET D001 0
SET I005 1
SET I000 10
DOUT OT#(1) OFF
MOVJ P0033 VJ=23.00
MOVJ P001 VJ=23.00
JUMP *LABEL2 IF I000=10
IMOV P0036 V=23.00 TF
*LABEL2
ADD P0036 P0043
DOUT OT#(1) ON
TIMER T=1.00
MOV... ...VJ=23.00
```

Fig. 19. A snippet of a robot program generated for a *Motoman* robot (INFORM).

3.8. Software interface

The developed software interface makes the link between the user, the CAD package and the robot. The functionalities and global architecture of the proposed software interface are schematically shown in

Fig. 20. This software interface runs under *Microsoft Windows* operating systems (*XP* or above) and in any industrial or personal computers with processing and graphical capacity to host *Autodesk Inventor*, Fig. 21. It was mainly written in VB.

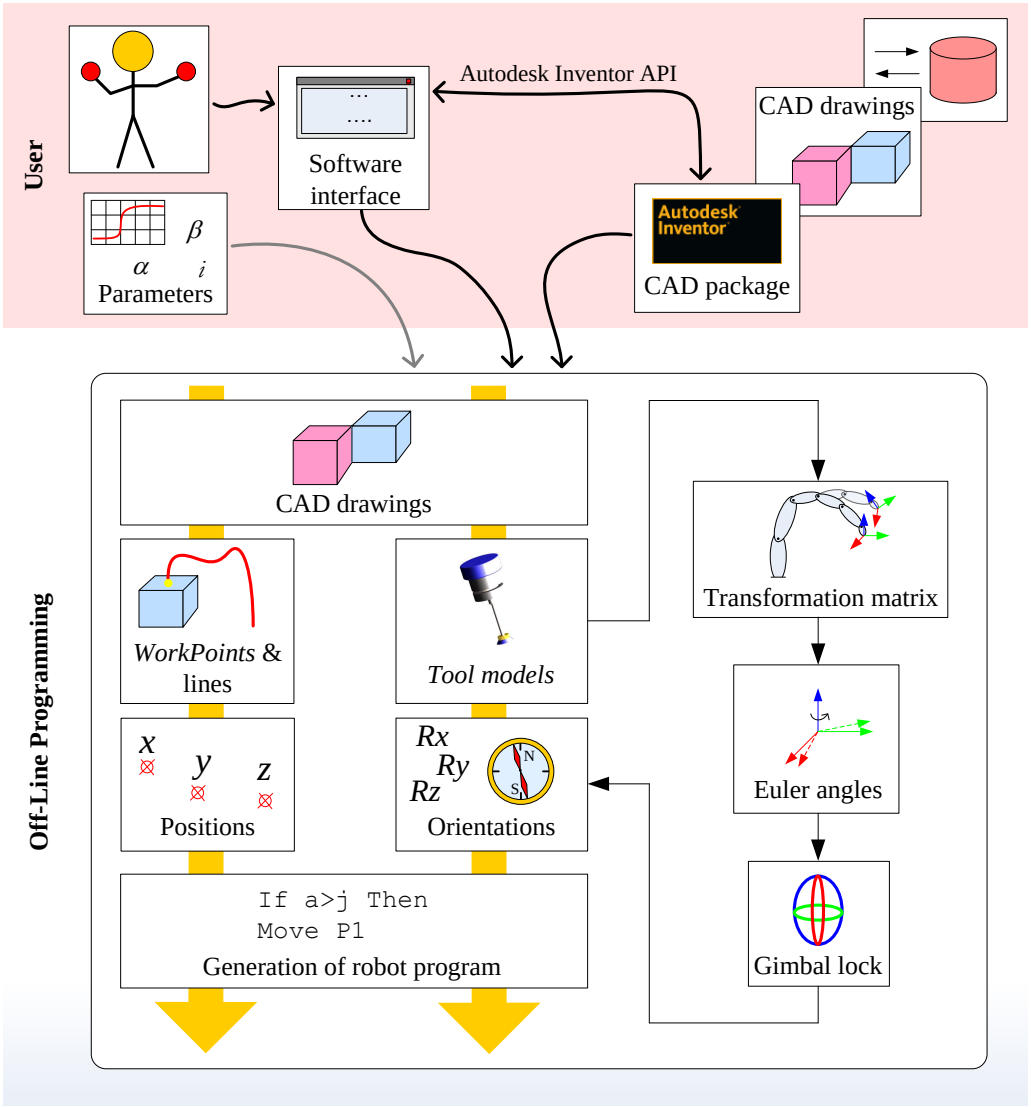


Fig. 20. Functionalities and architecture.

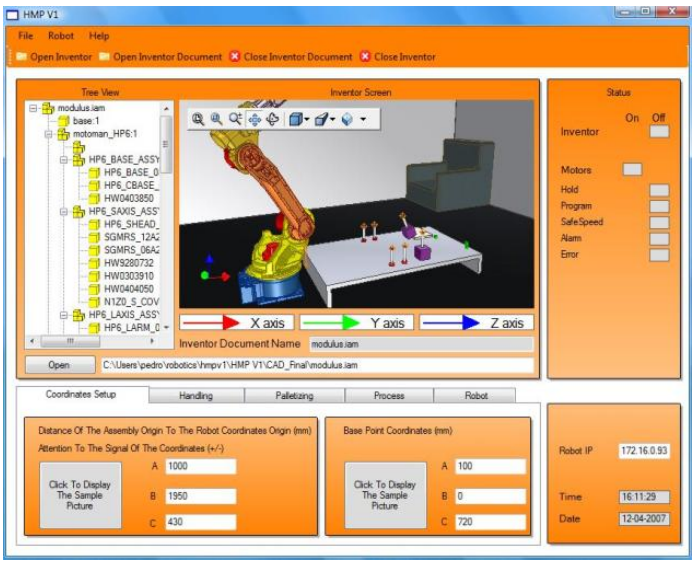


Fig. 21. Graphical user interface of the developed software.

4. Experiments

The CAD-based OLP system was validated in two different experimental setups, both representing practical scenarios of application of robots (manipulation tasks). The first experiment involves a robot manipulating objects from a location to another one and the second experiment a robot transporting an object between obstacles.

4.1. Experiment I

This experimental setup was designed to accomplish a simple object manipulation task. Robot programs are generated from a CAD *assembly model* of the robotic cell in study, Fig. 22, where simplified robot tool models represent the target poses for robot motion (initial poses and target poses). The robot task from which a robot program is generated consists in having the robot handling three objects from an initial to a final pose, Fig. 23 [49].

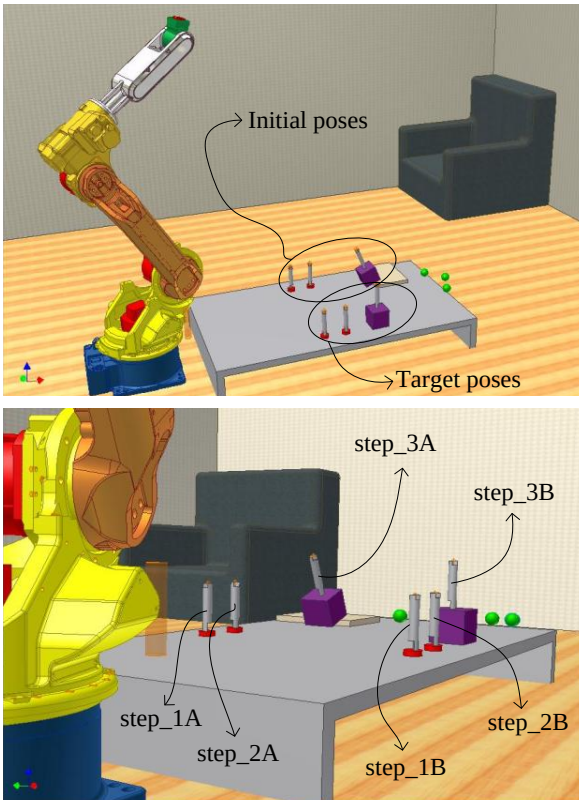


Fig. 22. Two different perspectives of the CAD *assembly model*.

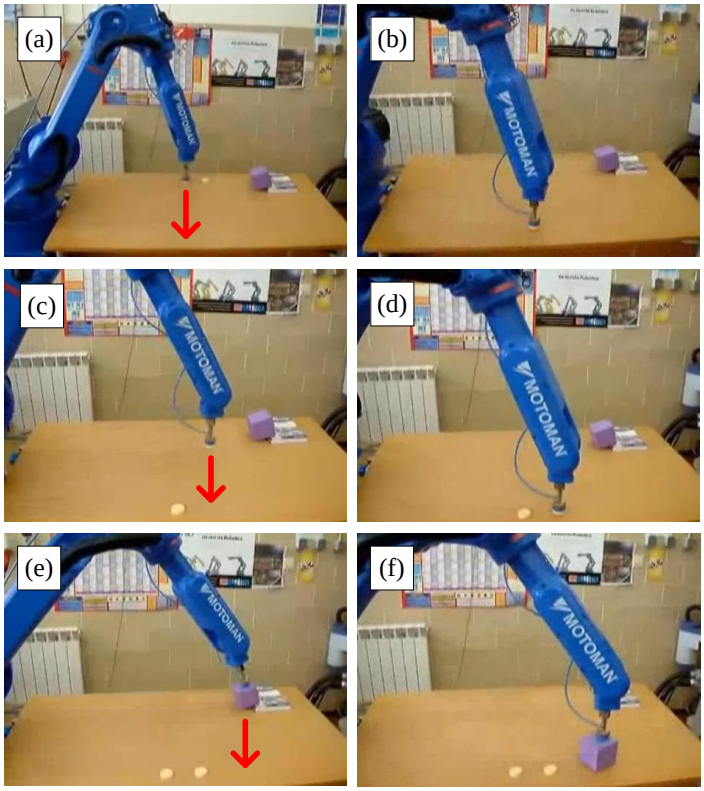


Fig. 23. Robot running the program generated from CAD.

4.2. Experiment II

In this experiment, robot programs are generated from a CAD *assembly model* in which the virtual paths (positional data) are represented in the form of straight lines, arcs and splines. The end-effector orientation is defined by placing tool models along the above mentioned virtual paths. These models define the orientation of the robot end-effector in each segment of the path, Fig. 24. The robot program generated from CAD is tested in a real scenario. As shown in Fig. 25 the real robot performs the manipulation task with success bypassing the obstacles without hitting them [50].

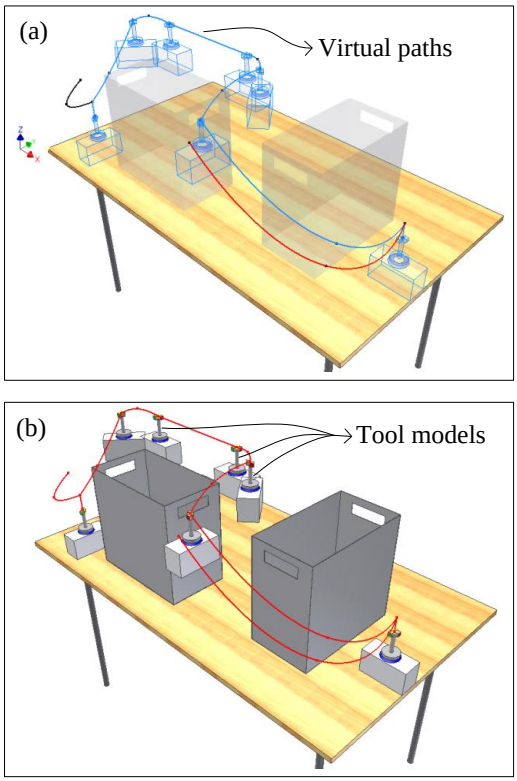


Fig. 24. CAD *assembly model* of the cell in study: with obstacles in invisible mode (a) and in visible mode (b).

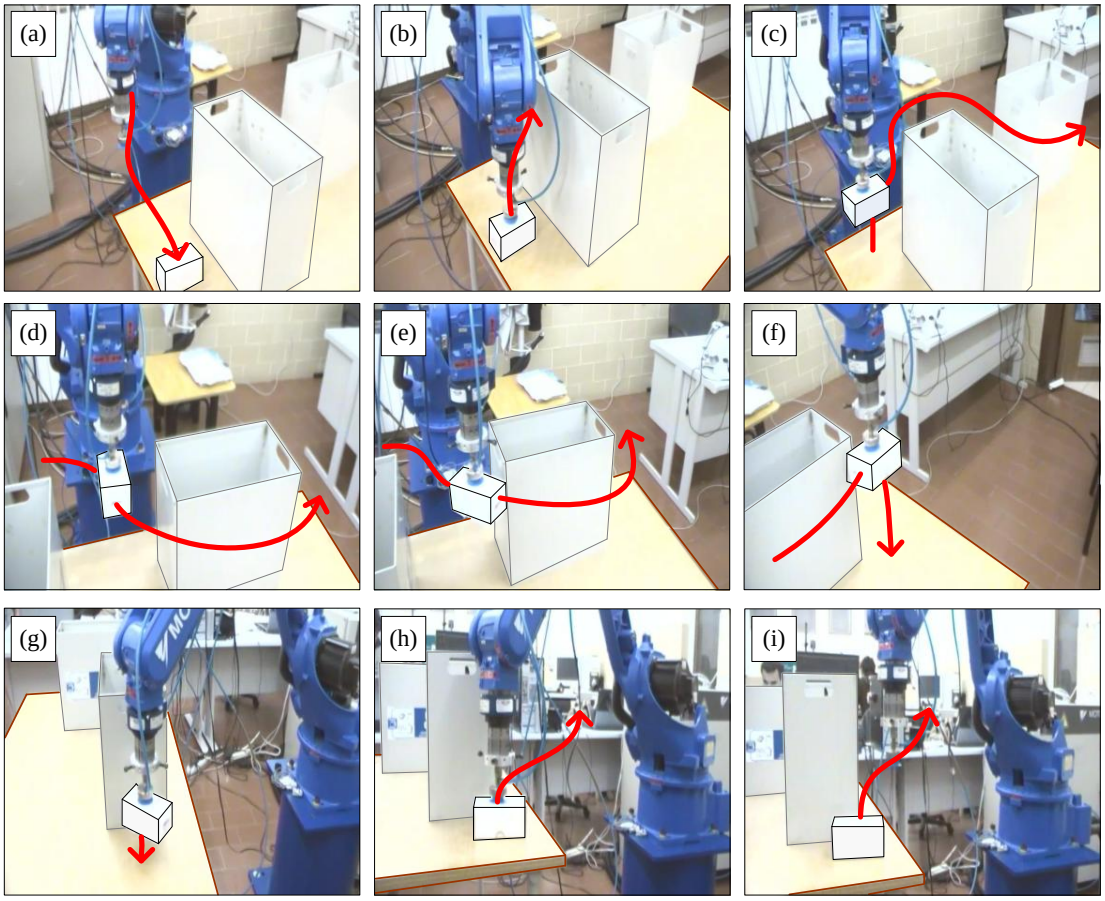


Fig. 25. Robot running the program generated from CAD.

4.3. Results and discussion

The experiments demonstrated the versatility of the proposed CAD-based OLP system. Robotic cell design and robot programming are embedded in the same interface and work through the same platform, *Autodesk Inventor*, without compatibility issues. In terms of accuracy, as in the case of commercial OLP software, the error that may exist comes from the robot and/or task calibration process and inaccuracies in the construction of the CAD models. In fact, error is always present, which may or may not be acceptable, depending on their magnitude and application under consideration. Often, task calibration errors arise from the little time and attention devoted to the calibration process. This situation is increasingly common as companies are constantly being asked to change production for new products. The above is true for all the robot programming and simulation systems based on virtual representation of objects in space, OLP. It is also important to note that after generating a robot program, it should be simulated in order to better visualize the robotic process (robot motion, possible collisions, the re-grasping operations, kinematic singularities, robot joint limits). Moreover, in order to be cautious with respect to possible errors, the robot programs generated off-line have to be tested (and adjusted if necessary) in the real robot (in the shop floor).

The proposed CAD-based HRI system is not the definitive solution for OLP. Nevertheless, it is an original contribution to the field, with pros and cons. The proposed system is limited in some aspects, for example in the level of sophistication and ability to generalize from particular situations. On the other

hand, the intuitiveness of use, short learning curve and the low-cost nature of the system appear as positive aspects, making it more accessible than common OLP software. All of these characteristics are fundamental when the objective is to spread the utilization of this kind of systems in SMEs or use it for educational and training purposes.

5. Conclusions and future work

A novel CAD-based OLP platform has been presented. Robotic cell design and OLP are embedded in the same interface and work through the same platform, a common commercial CAD package. It was proposed a method to extract robot paths (positions and orientations) from a CAD drawing of a given robotic cell. Such data are then treated and transformed into robot programs. In addition, the experiments showed that the proposed system is intuitive to use and has a short learning curve, allowing user with basic knowledge in robotics and CAD to create robot programs in just few minutes. In terms of accuracy, the error that may exist in the processes of OLP comes from inaccuracies in the robot/task calibration processes inherent to OLP and from situations where the CAD models do not reproduce properly the real robotic environment.

There are some aspects of the proposed CAD-based solution that can be improved in future. One aspect has to do with the algorithm to generate code, it has to be more generalist, flexible and easier to tune. An idea for future work is to have a graphical- or icon-based interface to tune the algorithm to generate code in a more intuitive way and independently of the robot language. The other aspect has to do with the existing error in the process. External sensing (force sensing for example) can help to deal with this situation by increasing the accuracy of the processes, making it less susceptible to error and simpler.

References

[1] Forge S, Blackman C. A helping hand for Europe: the competitive outlook for the EU robotics industry. Publication office of the European Union, 2010.

[2] Qi L, Zhang D, Zhang J, Li J. A lead-through robot programming approach using a 6-DOF wire-based motion tracking device. In: Proceedings of the 2009 IEEE International Conference on Robotics and Biomimetics, 2009, pp. 1773–1777.

[3] Hein B, Hensel M, Wörn H. Intuitive and model-based on-line programming of industrial robots: a modular on-line programming environment. In: Proceedings of the 2008 IEEE International Conference on Robotics and Automation, 2008, pp. 3952–3957.

[4] Hein B, Wörn H. Intuitive and model-based on-line programming of industrial robots: new input devices. In: Proceedings of the 2009 IEEE International Conference on Intelligent Robots and Systems, 2009, pp. 3064–3069.

[5] Bottazzi VS, Fonseca JFC. Off-line robot programming framework. In: Proceedings of the Joint International Conference on Automatic and Autonomous Systems and International Conference on Networking and Services, 2005, pp. 71–76.

[6] Mitsi S, Bouzakis KD, Mansour G, Sagris D, Maliaris G. Off-line programming of an industrial robot for manufacturing. The International Journal of Advanced Manufacturing Technology, 2004;26(3):262–267.

[7] Zha XF, Du H. Generation and simulation of robot trajectories in a virtual CAD- based off-line programming environment. The International Journal of Advanced Manufacturing Technology, 2001;17(8):610–624.

[8] Pan Z, Polden J, Larkin N, Duin SV, Norrish J. Recent progress on programming methods for industrial robots. Robotics and Computer Integrated Manufacturing, 2012;28(2):87–94.

[9] Freund E, Rokossa D, RoBmann J. Process-oriented approach to an efficient Off-line Programming of Industrial Robots. In: Proceeding of the 24th Annual Conference of the IEEE Industrial Electronics Society, 1998, pp. 208–213.

[10] Beyer L, Wulfsberg J. Practical robot calibration with ROSY. Robotica, 2004;22(5):505–512.

[11] Nubiola A, Bonev IA. Absolute calibration of an ABB IRB 1600 robot using laser tracker. Robotics and Computer Integrated Manufacturing, 2013;29(1):236–245.

[12] Muelaner J, Wang Z, Maropoulos P. Concepts for and analysis of a high accuracy and high capacity (HAHC) aerospace robot. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 2011;255(8):1393–1399.

[13] Kasik DJ, Buxton W, Ferguson DR. Ten CAD challenges. Computer Graphics and Applications, 2005;25(2):81–92.

[14] Bhanu B. CAD-based robot vision. IEEE Computer, 1987;20(8):12–16.

[15] Lin CI, Lu TF. CAD-based intelligent robot workcell. In: Proceedings of the 3rd International Conference on Computer Integrated Manufacturing, 1995, pp. 437–444.

[16] Neto P, Mendes N, Araújo R, Pires JN, Moreira AP. High-level robot programming based on CAD: dealing with unpredictable environments. Industrial Robot, 2012;39(3):294–303.

[17] Ferreira M, Moreira AP, Neto P. A low-cost laser scanning solution for flexible robotic cells: spray coating. The International Journal of Advanced Manufacturing Technology, 2012;58(9):1031–1041.

[18] Neto P, Pires JN, Moreira AP. CAD-based off-line robot programming. In: Proceedings of the 4th IEEE International Conference on Robotics, Automation and Mechatronics, 2010, pp. 516–521.

[19] Chen H, Fuhlbrigge T, Li X. A review of CAD-based robot path planning for spray painting. Industrial Robot, 2009;36(1):45–50.

[20] Sheng W, Xi N, Song M, Chen Y, MacNeille P. Automated CAD-Guided Robot Path Planning for Spray Painting of Compound Surfaces. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000, pp. 1918–1923.

[21] Arikan MAS, Balkan T. Process modeling, simulation, and paint thickness measurement for robotic spray painting. Journal of Robotic Systems, 2000;17(9):479–494.

[22] Chen H, Sheng W, Xi N, Song M, Chen Y. CAD-based automated robot trajectory planning for spray painting of free-form surfaces. Industrial Robot, 2002;29(5):426–433.

[23] Kim JY. CAD-based automated robot programming in adhesive spray systems for shoe outsoles and uppers. Journal of Robotic Systems, 2004;21(11):625–634.

[24] Schaefer T, Schraft D. Incremental sheet metal forming by industrial robot. Rapid Prototyping Journal, 2005;11(5):278–286.

[25] Pulkkinen T, Heikkilä T, Sallinen M, Kivikunnas S, Salmi T. 2D CAD based robot programming for processing metal profiles in short series manufacturing. In: International Conference on Control, Automation and Systems, 2008, pp.156–162.

[26] Nagata F, Hase T, Haga Z, Omoto M, Watanabe K. CAD/CAM-based position/force controller for a mold polishing robot. Mechatronics, 2007;17(4/5):207–216.

[27] Nagata F, Yoshitake S, Otsuka A, Watanabe K, Habib MK. Development of CAM system based on industrial robotic servo controller without using robot language. Robotics and Computer-Integrated Manufacturing, 2013;29(2):454–462.

[28] Nagata F, Yoshitake S, Otsuka A, Watanabe K, Habib MK. CAM system without using robot language for an industrial robot RV1A. In: Proceedings of the 2012 IEEE International Symposium on Industrial Electronics, 2012, pp. 1529–1534.

[29] Cerit E, Lazoglu I. A CAM-based path generation method for rapid prototyping applications. The International Journal of Advanced Manufacturing Technology, 2011;56(1/4):319–327.

[30] Feng-yun L, Tian-sheng L. Development of a robot system for complex surfaces polishing based on CL data. The International Journal of Advanced Manufacturing Technology, 2005;26:1132–1137.

[31] Chen YH, Hu YN. Implementation of a robot system for sculptured surface cutting - Part 1 - Rough machining. International Journal Advanced Manufacturing Technology, 1999;15(9):624–629.

[32] Hu YN, Chen YH. Implementation of a robot system for sculptured surface cutting - Part 2 - Finish machining. International Journal Advanced Manufacturing Technology, 1999;15(9):630–639.

[33] Murphy K, Norcross R, Proctor F. CAD directed robotic deburring. In: 2nd International Symposium on Robotics and Manufacturing Research, Education, and Applications, 1988.

[34] Sallinen M, Sirviö M. Robotized system for prototype manufacturing of castings and billets. In: Arai E. and Arai T. (Eds.): Mechatronics for Safety, Security and Dependability in a New Era, 2006, pp. 277–280, Elsevier, Oxford.

[35] Murarka A, Kuipers B. Using CAD drawings for robot navigation. In: IEEE Systems, Man and Cybernetics Conference, 2001, pp. 678–683.

[36] Wang QH, Li JR, Wu BL, Zhang XM. Live parametric design modifications in CAD-linked virtual environment. The International Journal of Advanced Manufacturing Technology, 2010;50(9):859–869.

[37] Neto P, Pires JN, Moreira AP. Robot simulation: an approach based on CAD – Autodesk Inventor. IEEE Robotics and Automation Magazine, submitted.

[38] Yoo TK, Lee WH. Blend Shape with Quaternions, In: International Conference on Convergence Information Technology, 2007, pp. 776–780.

[39] Pollard N, Hodgins JK, Riley MJ, Atkeson C. Adapting human motion for the control of a humanoid robot. In: IEEE International Conference on Robotics and Automation, 2002, pp. 1390–1397.

[40] Mendes N, Neto P, Pires JN, Loureiro A. Discretization and fitting of nominal data for autonomous robots. Expert Systems with Applications, 2013;40(4):1143–1151.

[41] Wang HF, Zhang YL. CAD/CAM integrated system in collaborative development environment. Robotics and Computer-Integrated Manufacturing, 2002;18(2):135–145.

[42] Liang M, Ahamed S, van den Berg B. A STEP based tool path generation system for rough machining of planar surfaces. Computers in Industry, 1996;32(2):219–231.

[43] Xu XW, He Q. Striving for a total integration of CAD, CAPP, CAM and CNC. Robotics and Computer-Integrated Manufacturing, 2004;20(2):101–109.

[44] Xu XW. Realization of STEP-NC enabled machining. Robotics and Computer-Integrated Manufacturing, 2006;22(2):144–153.

[45] Liu Z, Bu W, Tan J. Motion Navigation for Arc Welding Robots Based on Feature Mapping in a Simulation Environment. Robotics and Computer-Integrated Manufacturing, 2010;26(2):137–144.

[46] Freund E, Luedemann-Ravit B. A system to automate the generation of program variants for industrial robot applications. In: IEEE/RSJ International Conference on Intelligent Robots and System, 2002, pp. 1856–1861.

[47] Chen H, Sheng W. Transformative CAD based industrial robot program generation. Robotics and Computer-Integrated Manufacturing, 2011;27(5):942–948.

[48] Freund E, Luedemann-Ravit B, Stern O, Koch T. Creating the architecture of a translator framework for robot programming languages. In: IEEE International Conference on Robotics and Automation, 2001, pp. 187–192.

[49] Available: http://www2.dem.uc.pt/pedro.neto/Video_2012_1.wmv

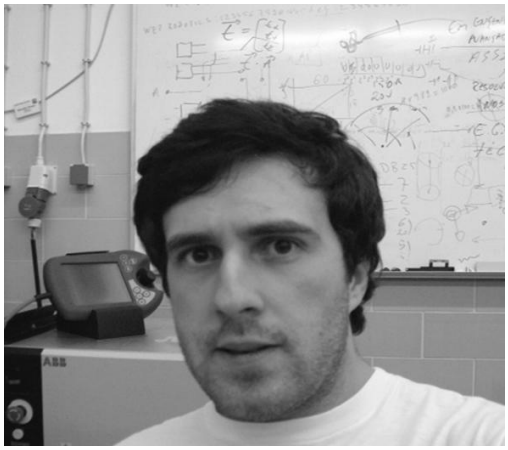
[50] Available: http://www2.dem.uc.pt/pedro.neto/Video_2012_2.wmv

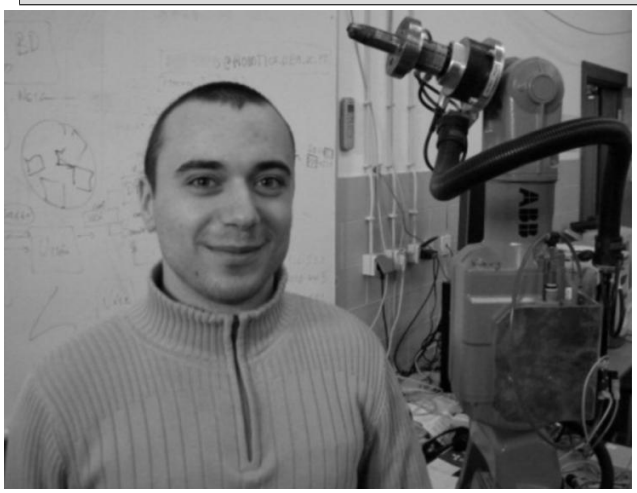
Biography Pedro Neto

Pedro Neto was born in Coimbra, Portugal, on February 1, 1984. He received the Bachelor degree and Ph.D. degree in Mechanical Engineering from the University of Coimbra in 2007 and 2012, respectively. He has been involved in teaching activities since 2010 as Assistant Professor at the Department of Mechanical Engineering of the University of Coimbra. His research interests include: human-robot interaction, pattern recognition, CAD-based robotics and sensor fusion. Pedro Neto is author of several journal and conference publications. He participated in two European funded R&D projects, FP6 and FP7, and national projects.

Biography Nuno Mendes

Nuno Mendes is currently a Ph.D. student at the University of Coimbra. He received the Bachelor degree in Mechanical Engineering from the University of Coimbra in 2008. His research interests include: CAD-based robotics, sensor fusion, force control, Fuzzy and robotic friction stir welding. Nuno Mendes is author of several journal and conference publications.





Research highlights

- A novel CAD-based off-line robot programming solution;
- Robot programs are automatically generated from a CAD drawing;
- Robot cell design and robot programming are embedded in the same interface;
- The system is intuitive to use and presents a short learning curve.